

# COS 318: Operating Systems

## OS Structures and System Calls

Kai Li  
Computer Science Department  
Princeton University

(<http://www.cs.princeton.edu/courses/cos318/>)

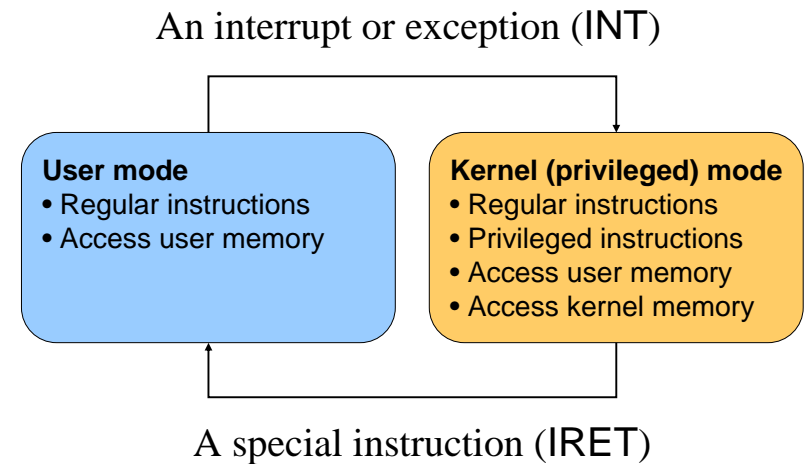
## Outline

- ◆ Protection mechanism
- ◆ OS structures
- ◆ System calls and library calls

## Protection Issues

- ◆ CPU
  - Prevent a user from using the CPU forever
- ◆ Memory
  - Prevent a user from accessing others' data
  - Prevent users from modifying kernel code and data structures
- ◆ I/O
  - Prevent users from performing illegal I/Os
- ◆ Question
  - What's the difference between protection and security?

## Architecture Support: Privileged Mode



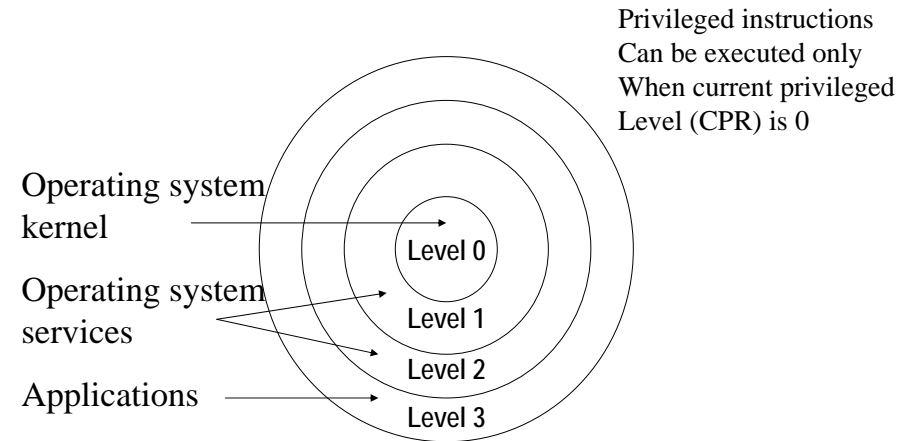
## Privileged Instruction Examples

- ◆ Memory address mapping
- ◆ Flush or invalidate data cache
- ◆ Invalidate TLB entries
- ◆ Load and read system registers
- ◆ Change processor modes from kernel to user
- ◆ Change the voltage and frequency of processor
- ◆ Halt a processor
- ◆ Reset a processor
- ◆ Perform I/O operations



5

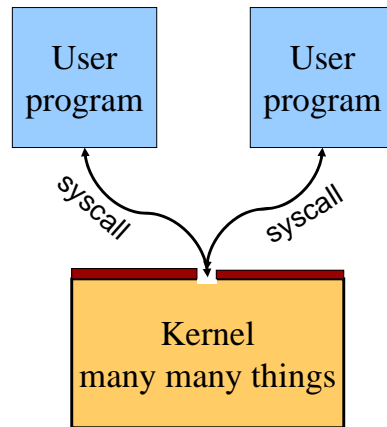
## x86 Protection Rings



6

## Monolithic

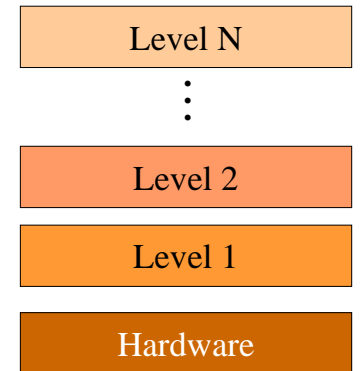
- ◆ All kernel routines are together
- ◆ A system call interface
- ◆ Examples:
  - Linux
  - BSD Unix
  - Windows
- ◆ Pros
  - Performance
  - Shared kernel space
- ◆ Cons
  - Stability
  - Flexibility



7

## Layered Structure

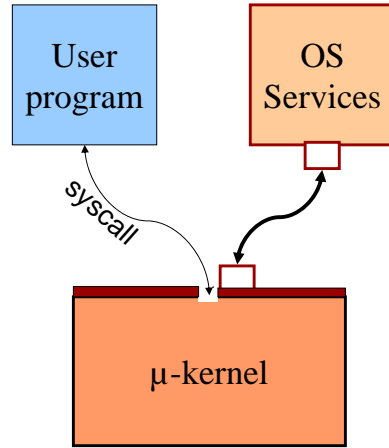
- ◆ Hiding information at each layer
- ◆ Layered dependency
- ◆ Examples
  - THE (6 layers)
  - MS-DOS (4 layers)
- ◆ Pros
  - Layered abstraction
- ◆ Cons
  - Efficiency



8

# Microkernel

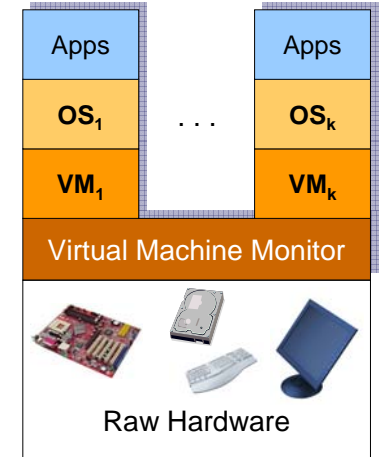
- ◆ Micro-kernel is “micro”
- ◆ Services are implemented as regular process
- ◆ Micro-kernel get services on behalf of users by messaging with the service processes
- ◆ Examples:
  - Taos, L4, Mach, OS-X
- ◆ Pros?
  - Easier to develop services
  - Fault isolation
  - Customization
- ◆ Cons?
  - Lots of boundary crossings
  - System call overhead



9

# Virtual Machine

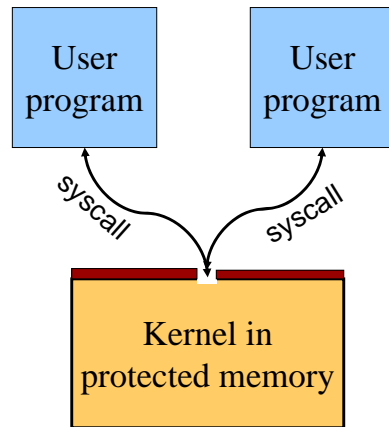
- ◆ Virtual machine monitor
  - Virtualize hardware
  - Run several OSes
  - Examples
    - IBM VM/370
    - Java
    - VMWare
- ◆ What would you use virtual machine for?
- ◆ Does virtual machine need more than two modes?



10

# System Call Mechanism

- ◆ Assumptions
  - User code can be arbitrary
  - User code cannot modify kernel memory
- ◆ Design Issues
  - Makes a system call with parameters
  - The call mechanism switches code to kernel mode
  - Execute system call
  - Return with results



11

# Interrupt and Exceptions

- ◆ Interrupt Sources
  - Hardware (by external devices)
  - Software: INT n
- ◆ Exceptions
  - Program error: faults, traps, and aborts
  - Software generated: INT 3
  - Machine-check exceptions
- ◆ See Intel document volume 3 for details

12

# Interrupt and Exceptions (1)

Vector #	Mnemonic	Description	Type
0	#DE	Divide error (by zero)	Fault
1	#DB	Debug	Fault/trap
2		NMI interrupt	Interrupt
3	#BP	Breakpoint	Trap
4	#OF	Overflow	Trap
5	#BR	BOUND range exceeded	Trap
6	#UD	Invalid opcode	Fault
7	#NM	Device not available	Fault
8	#DF	Double fault	Abort
9		Coprocessor segment overrun	Fault
10	#TS	Invalid TSS	

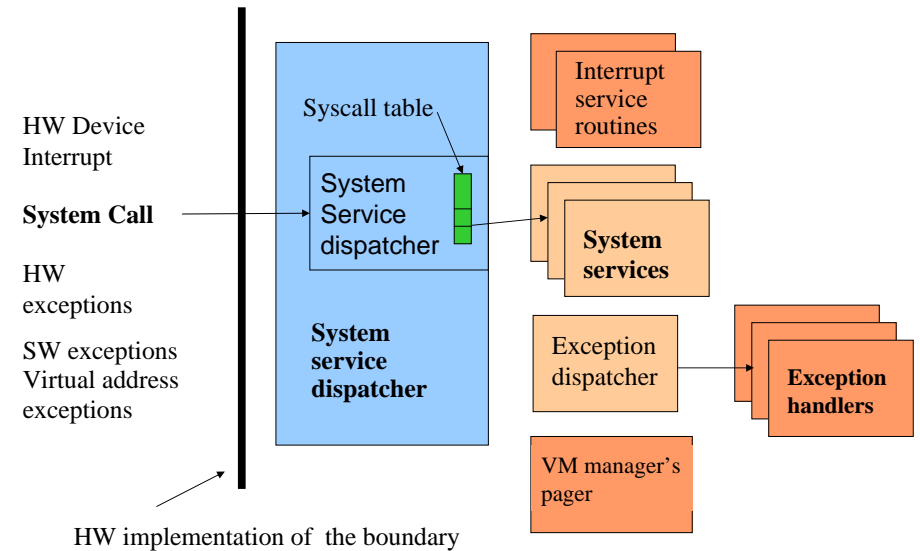
# Interrupt and Exceptions (2)

Vector #	Mnemonic	Description	Type
11	#NP	Segment not present	Fault
12	#SS	Stack-segment fault	Fault
13	#GP	General protection	Fault
14	#PF	Page fault	Fault
15		Reserved	Fault
16	#MF	Floating-point error (math fault)	Fault
17	#AC	Alignment check	Fault
18	#MC	Machine check	Abort
19-31		Reserved	
32-255		User defined	Interrupt

# System Calls

- ◆ Operating system API
  - Interface between an application and the operating system kernel
- ◆ Categories
  - Process management
  - Memory management
  - File management
  - Device management
  - Communication

# OS Kernel: Trap Handler



# Passing Parameters

- ◆ Pass by registers
  - # of registers
  - # of usable registers
  - # of parameters in system call
  - Spill/fill code in compiler
- ◆ Pass by a memory vector (list)
  - Single register for starting address
  - Vector in user's memory
- ◆ Pass by stack
  - Similar to the memory vector
  - Procedure call convention

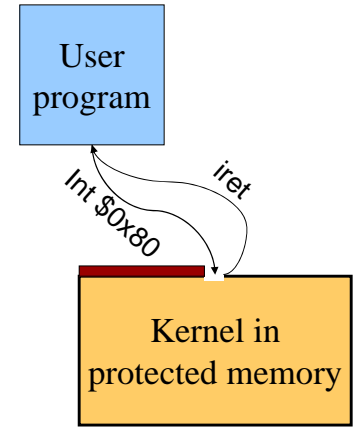


# Library Stubs for System Calls

```

    Example:
    int read( int fd, char * buf, int size)
    {
        move fd, buf, size to R1, R2,
        R3
        move READ to R0
        int $0x80
        move result to Rresult
    }
    
```

Linux: 80  
NT: 2E

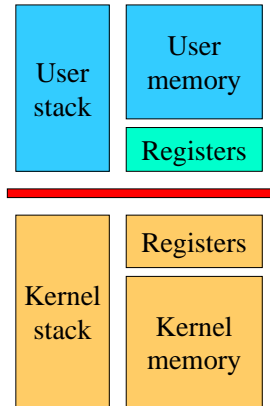


# System Call Entry Point

## EntryPoint:

- switch to kernel stack
- save context
- check R<sub>0</sub>
- call the real code pointed by R<sub>0</sub>
- place result in R<sub>result</sub>
- restore context
- switch to user stack
- iret (change to user mode and return)

(Assume passing parameters in registers)



# Other Design Issues

- ◆ System calls
  - There is one result register; what about more results?
  - How do we pass errors back to the caller?
  - Can user code lie?
  - How would you perform QA on system calls?
- ◆ System calls vs. library calls
  - What should be system calls?
  - What should be library calls?



## Division of Labors (or Separation Of Concerns)

### Memory management example

- ◆ Kernel
  - Allocates “pages” with hardware protection
  - Allocates a big chunk (many pages) to library
  - Does not care about small allocs
- ◆ Library
  - Provides malloc/free for allocation and deallocation
  - Application use these calls to manage memory at fine granularity
  - When reaching the end, library asks the kernel for more



21

## Feedback To The Program

- ◆ Applications view system calls and library calls as procedure calls
- ◆ What about OS to apps?
  - Various exceptional conditions
  - General information, like screen resize
- ◆ What mechanism would OS use for this?

Application

Operating System



22

## Summary

- ◆ Protection mechanism
  - Architecture support: two modes
  - Software traps (exceptions)
- ◆ OS structures
  - Monolithic, layered, microkernel and virtual machine
- ◆ System calls
  - Implementation
  - Design issues
  - Tradeoffs with library calls



23